Artificial Intelligence, Gene Therapy and Cancer

James Bonnar*
email: bonnarj@gmail.com

March 2025

Abstract

The emerging science of artificial intelligence (AI) can potentially help find new compounds to treat cancer. Progress in this area has been slow because many existing AI techniques are inappropriate for handling molecular structures and chemical reactions. Neural networks are a good example of an algorithm incapable of handling a wide variety of molecular structures, so an alternative method, Bonnar's algorithm, is developed here. It is interesting to explore how Bonnar's algorithm can be used to find new lead compounds for the treatment of cancer via gene therapy.

In this paper, because we are going to be using an AI technique designed for handling chemical reactions, we shall apply Bonnar's algorithm to find compounds capable of correcting a specific mutation in a cell's genome (reacting the lead compound in silico with a specific sequence of DNA, covalently changing the DNA). A gene known as TP53 is the most commonly mutated gene in cancer. It has long been known TP53 is a tumor-suppressor gene

*©2025 by James Bonnar. Anyone may copy and distribute this work. Any derivatives of this work must cite this article and the author.

and it is very-well studied. TP53 codes for the p53 protein. Many TP53 mutations are missense mutations and are located in several hotspots. Among these hotspot mutations, p53-R175H has the highest occurrence in diverse cancers. R175H means that the arginine in position 175 of p53 is changed to a histidine. This happens as a result of exchanging a CGC DNA codon, which codes for Arg, with a CAC DNA codon, which codes for His. Our goal is to unmutate the gene, i.e., A→G on the plus strand and T→C on the minus strand, and to do it at a specific site in the genome.

We will enact Bonnar's algorithm on the platform `Mathematica`, generate in silico libraries of compounds that follow a theme using a Markovian text generator, and (it is suggested) we use `Chemicalize` (available for free from ChemAxon) to draw our molecules, convert them to SMILES strings, and to view lead compound structures generated by Bonnar's algorithm from SMILES strings.

For the algorithm's training data, we make use of the SMILES Chemical Reaction Database, which one may download from this webpage. The SMILES Chemical Reaction Database is a set of files containing structural information about pairs of reactants and products of two million different chemical reactions. The simplified molecular-input line-entry system (SMILES) of representing molecular structures is used to represent molecular connectivity and stereochemical relationships as strings of characters, and indeed chemical reactions as well. The SMILES Reaction Database is now 186.8 MB in size, and it contains two million reactant-product pairs

extracted from thousands of respected journals and patents, contained in six files. The reaction data entries in each file of the database occur on consecutive lines of the file, which are delineated by newline characters.

So there are basically four pieces of software to familiarize oneself with:

1. `Mathematica` and the Wolfram programming language
 Note: One may use Mathematica for free by visiting https://www.wolframcloud.com
2. SMILES Chemical Reaction Database – downloadable from this webpage
3. Markovian text generator – downloadable from this webpage
4. `Chemicalize` – https://chemicalize.com

Introduction

There are technical and non-technical (psychological and sociological) reasons science has been unable to find cures for cancer and heritable diseases, despite the exponential increase in knowledge of disease mechanisms we currently witness. None of the reasons are valid and essentially amount to excuses.

This work is the second published report on the initial results of a long-term project originally conceived over 25 years ago. At that time, I was a student of chemistry and physics at the University of Wisconsin-Parkside, near the completion

of my degree. An involvement with an independent study research course in the physics department dealing with discrepancies in how Bohr's correspondence principle predicts highly-energized (Rydberg) atoms should behave and their actual chaotic behavior (quantum chaos) provided my first encounter with the unknown and the insufficiency of fundamental theory to efficiently model complex systems. To my surprise, that first experience with ineffective or incomplete scientific theory and practice radically undermined most of my basic conceptions about the completeness, capabilities, validity and practice of contemporary science and the reasons for both its successes and failures.

Those conceptions were practical – I shared the same perspective on science that an engineer probably would. In this, I mean I took a constructive utilitarian point of view, rather than an analytical point of view on the ultimate objectives of science. The goal of any scientific endeavor, for me, was to ultimately be able to control, alter or build with the object of study. This instinct has deep roots in American culture and comes quite naturally to a creature possessing an opposable thumb. The results of this value system was a gradual change in my career plans over the course of my education – from premed to biochemistry to mathematics and physics. The work presented in this paper is a reintegration of everything I learned.

My first opportunity to vigorously pursue the ideas set forth in this paper came with the development of `Mathematica 7`, but were not successful until the advent of the machine learning algorithms present in `Mathematica 10`. Many wrong avenues were taken along the way. Without the freedom from low-level programming tasks that `Mathematica` pro-

vides, the development of Bonnar's algorithm would have been far more difficult and most likely would not have been achieved. Much of my time in previous years was spent on the study of programmming proper. In particular, I continued to program in C and Java, quite laboriously reproducing the algorithms I wished to use to answer a single question, if in fact that question could even be answered using that algorithm in practice. However, with the addition of `Mathematica` to my repertoire, my research has progressed at a rate 100-fold quicker. Stephan Wolfram, the original creator of `Mathematica`, who graduated from Caltech with his doctorate in theoretical physics at the age of nineteen, is the greatest single contributor to modern computer mathematics.

Far more than other recent scholars, Wolfram and his group has shown that a bold reconsideration of the primitives of science can be quite beneficial, though we now live in a period of rigid thought in the theoretical sciences, despite the fact that technology has become very progressive. The theoretical sciences have unfortunately become mired with politics, elitism and endeavors totally unsupported by experimental evidence.

To recap, much of my time in previous years was spent exploring fields without apparent relation to biomedical science, but in which the totality of research could be unified and generalized – the importance of which history is now bringing to light. It has become deeply set into the social order of science *not* to do this. Only specialization is rewarded or respected, and being a generalist can be misconstrued as having a lack of direction. The adage "no good deed goes unpunished" applies in the scientific world.

Fortunately, the ideas I thereby assimilated oriented my research and provided a scaffold for most of my more advanced thinking. The same orientation and scaffold gave a unity and direction to my thoughts in all of my research. Even further, my work is a direct expression of my subconscious machinery at work. Quite literally, some of my ideas and solutions came to me *in my dreams*. These forces always play a role in truly creative scientific research. Others are a testament to the way in which new experimental or computational technology may help a researcher overcome an incompatible theory. Experimental technology has a long history of inducing the formation of new theories. New computational technology will do the same in the sense that it allows scientists to seriously entertain more complex theories without the subconscious fear of not being able to do anything with the theory. In this way, my work chronicles the emergence of a new theoretical framework.

An early solidifying experience in the development of my career was the experience of being berated by a mathematics professor for not pursuing the mathematical approach to science very early in my academic career. The experience made a lasting impression on me. There exists very different attitudes about how science should be done in the non-mathematical versus the mathematical sciences (by non-mathematical sciences I mean biology and most of chemistry, and by mathematical sciences I mean physics, engineering, computer science and mathematics itself). The number and extent of disagreements between these two groups concerning the nature of true science and how it should be done is surprising. But history forces me to doubt that the mathematics-based natural sciences are any more legitimate or permanent in their conclusions than the non-mathematical

sciences are. It is not a good thing to get overly impressed by the existence of a mathematical model to describe a theory. It does not necessarily impart truth to the theory. Mathematics can model anything; whether we assign true or false meaning to the equations is a matter of interpretation.

Yet, somehow, the practice of physics and engineering fail to evoke the same frustration over fundamental objectives that are endemic among fields such as cancer research or chemical synthesis. Meditating upon the source of that difference in these two communities led me to the realization that learned roles play a dominant part in scientific research. These are universally-recognized (and enforced) modes of operation that provide not only a subset of admissible problem domains to a group of practitioners, but also a limited subset of admissible *solution domains* to that group. Biologists don't typically scribble partial differential equations on the chalkboard when discussing gene expression, nor do chemists talk about the latest algorithmic advancements in numerical analysis when discussing molecular dynamics calculations. Molecular biologists attempt gene therapy with molecular biology tools (virus vectors). Chemists treat disease with small organic molecules relatively easy to synthesize. These seem like reasonable modes of operation only because cultural expectation allows for them and traditions demand them. Once this realization occurred, my research direction was legitimized and justified in my mind, and a new outlook emerged.

Since my most important objective is to change the way familiar systems are evaluated, the occassional sketchiness in this paper is no drawback. I want the readers to use their imaginations. Chance favors the prepared mind, as the

saying goes. If the reader's own frame of mind is open to the sort of suggestions given, he or she may find the material much easier to learn and digest, and improve upon.

The take on science developed in my research suggests several new avenues of investigation which I'm convinced will prove fruitful. And the manner in which unexpected results occurred has gained my attention – each of these results merits further detailed study. In my view, every scientific discovery worth publishing alters the perspective of the person reading about it. Then that change of perspective itself should have an effect upon the content of future publications and research.

A New Light

What I would like to do in this section is elucidate a bit the scientific structure I spent the last twenty-five years struggling to build – a structure which in the end proved relatively simple to implement. It is a way of thinking which has some very exciting implications both now and well into the future. I had gotten interested in the semantics of representations (assignment of meaning to syntax) and had also been thinking about the equivalence of different representations in physics. For example, the equivalence of matrix and wave mechanics. Equally important, I had also been studying the correspondence principle and had been meditating a great deal on the implications of this phenomena in terms of computational efficiency. Newtonian calculations on large scale objects are far more efficient than sum total quantum mechanical calculations would be on the object's parts.

Even orthodox viewpoints often demonstrate the fact that simplicity can often be found even in the most complex phenomena of nature. What is unorthodox about my viewpoint is the belief that simplicity can *always* be found in the complex, if one is willing to see things in a new light. From the high-level differential equations that describe turbulent fluid flows, to the wave mechanics describing electron behavior about the nucleus, these are all examples of simplicity found in phenomena which could be treated at far more complicated deeper levels. Deepening a theory brings increased understanding, but not necessarily an ability to compute things better. There is actually merit in looking for what one may think of as more superficial theories for the purposes of doing computations efficiently. Two theories which say very different things in general may say essentially the same thing, they are equivalent, in some limited domain. I believe I have found a way to derive computationally-efficient, secondary theories to model phenomena already explained by computationally-inefficient primary theories within certain limited problem domains. Specifically, we shall be extracting the knowledge inherent in a chemical reaction database and applying it to make predictions about reactions involving complex molecular species, namely, the determination of molecules to react with a DNA sequence to transform the sequence in some desired way – this process is called Bonnar's algorithm. What we will compute is in effect a necessary but not-necessarily-sufficient condition that a proposed molecule would produce the transformed sequence when reacted with the initial sequence of DNA. This allows us to tremendously narrow down our search of chemical space for the proper reactant in our effort to correct a gene.

It is a common misconception that the study of complex, deep phenomena such as say, quantum field theory, will

enable one to easily figure out what is going on with higher-level systems such as say, a methane molecule. It seems reasonable – the whole is just the sum of its parts, right? But when one tries to actually do such a thing, things rarely work out, except in the simplest of cases. Reductionism has its benefits, but putting something back together is not the mirror inverse of tearing it apart, especially in regards to computation on complex microscopic systems such as biomolecules. Gradually, I began to realize that there is a fundamental problem with the whole approach science is using to describe such systems. It is commonly believed that to describe such systems, one must build from the bottom up (but how many chemists would use quantum field theory to describe the hydrogen molecule?). Secondly, I have found experimentally that it is more productive, computationally-efficient theories can always be found, if one uses *ground-level theories*. By ground-level theory, I mean not a lower-level or higher-level theory, but a theory that employs data at the level about which the theory is meant to make predictions. This is no way minimizes the importance of microscopic descriptions of matter, but what I mean by this is that it is more effective to use molecular structure itself as the data from which we build our theories about the behavior of molecular structures. It is *not* necessary to build up from say, mathematical models intended to describe the electrons within atoms, to describe say, a protein's catalytic behavior (at least as far as computing its behavior).

But one may be asking how we are to find such theories. The quantum theory of molecular structure itself was so hard won. How are we to find these new ground-level theories? The good news is that it is not only possible, but the process of finding these mathematical models has been automated

and is very easily implemented. Computationally-efficient theories of chemical reactivity can and have been found using nearest-neighbors analysis of a database of SMILES strings that describe chemical reactions, but I will discuss more on the later.

If one looks at history, the use of mathematics to describe natural phenomena has been a defining feature of the advanced sciences. It has been observed that the more advanced the science, the more mathematical it becomes. Even in biology, mathematics is making an inroad in the form of data analysis or what has come to be called "bioinformatics". But despite the existence of subfields such as "physical chemistry", one may be surprised to learn that such subfields often have little to do with the practice of the rest of the field. For example, organic chemists deal with synthesizing molecules of sufficient complexity that physical chemists have little chance to add anything helpful to the organic chemist's work in practice. The theoretical does provide understanding, but is often of little to no predictive value because the computations cannot be done.

Imagine an alternate universe in which the equations of electromagnetism were so inscrutable to numerical computation that electrical engineers could not do reliable computations concerning systems any more complex that the common light bulb. In this world, engineers could analyze and understand even the most complex supercomputers found laying around, but could not build a radio if their lives depended on it. At the finest universities, courses would be taught on subjects such as "combinatorial electronics", in which students would build libraries of random appliances and develop assays to detect good appliances. These are precisely the strange

set of circumstances under which chemists must live. They must do so because the equations they have been taught *must* be used to do legitimate, acceptable computations on chemical systems are not amenable to accurate numerical computation, especially in regards to complex systems. This is a result of conditioning and social pressure, not scientific necessity.

Is there someway to go beyond the accepted paradigm in thinking about complex systems? What are the necessary ingredients of a computation concerning a system we know is composed of "parts"? How is it that nature can form complex systems with equal ease as it forms simple systems? In looking for the ingredients or "primitives" of an efficient computation, might we also be inadvertently also be learning something about nature itself? It was discovered many years ago by Richard Feynman that the movement of subatomic particles could be accounted for if, in moving from place to place, the particle was considered to take *all* possible paths. These paths add up, but only some of them add to the sum significantly. So nature itself, even at its deepest levels, displays this tendency of computational efficiency. The crucial thing to understand is that these primitives do not need to be based on the smallest parts of the complex system.

But if one is to do computations on natural systems at all, then one is going to have to implicitly assume that nature follows definite rules, which appears to be the case. But exactly who is define what the *complete* set of rules are for us? In nature, there are rules, but then there are rules governing the rules, and then there are rules governing the rules that govern the rules, and so forth. The classical laws of physics must be satisfied by quantum mechanics in the limit

of large numbers, the laws of ordinary quantum mechanics must be satisfied by string theory, and so on. Is there any way to make these observations systematic and useful? In the past, there was no way to do so, but now we have computers. What sort of computer program is relevant to the determination of computational primitives? In programming we are used to developing long, complicated programs suited to a particular task. But here we need a way to derive, in general, the efficient computational primitives of a problem domain, in particular those of chemical reactivity. Such an algorithm was implemented by the author and is presented in this paper – it is a machine learning algorithm, a form of *string transfomation learning*, for equating ordered pairs of character strings (representing reactants and products) to equal arbitrary integers, deriving just what the rule or map is governing the relation between the ordered pairs in general, and then using this rule to make predictions on novel cases in the domain of the map. In this scheme the strings of characters represent molecules. I take creative license and coin the algorithm for finding reactants that transform an initial strand of DNA to some desired product strand of DNA *Bonnar's algorithm.*

One will see that here we have a systematic method for finding the computational primitives for complex molecular systems and this provides a way of doing all the computer experiments we shall have to do in an efficient way. For many years, developing a method for finding these computational primitives became an obsession. As Linus Pauling said, the best way to get good ideas is to get a lot of ideas. Finally, the method I found has developed to the point where we can start looking at the questions in science, particularly in medicinal and organic chemistry and gene therapy, that we

want to. I feel very much like I am using a tool that can be simply pointed at a problem and immediately see the phenomenon in a new light, a computationally-efficient one at that.

Very simple rules can give incredibly complicated, even surprising behavior. It is a robust, general phenomenon. How come a more fundamental approach such as this hadn't been in use in the molecular sciences for decades? One reason is, until recently, high-speed computers were not widely available. One can see these things only by doing many computer experiments. With our ordinary intuition alone, there just did not seem to be any reason to even try these computer experiments. It seemed so obvious that they wouldn't show anything interesting, at the time. We were wrong. But now, we can see hints of this phenomenon from the past, even in mathematics itself. As an example, the sequence of primes, irregular and able to pass any test of randomness we have, can be generated from a simple deterministic formula. Similarly, the digits of π can also be generated from a deterministic formula, though they can also pass any test of randomness. In the same way, nature itself seems to produce so much that seems, to us, so complicated. But complexity is really only a matter of perspective. It is as if nature has some secret for building complex systems.

Many scientists believe that in order to create things we have to operate under the constraint that we have to foresee what the things we create are going to do. So many have forced themselves into a limiting solution space of special algorithms which only have predictable, foreseeable behavior. Presumably, nature is under no such constraint, in that way somehow producing the degree of complexity we

humans fail to emulate in our creations. Using the protocols of string transformation learning, it is now possible to synthesize molecules and predict reactions of extremely complex molecules. As Richard Feynman wrote, "What I cannot create, I do not understand.", which was written on his chalkboard at the time of his death in 1988.

Some scientists may feel uncomfortable with using these new ground-level models when they tackle a problem. Some have a tendency to seek the approval of others in how they go about solving a problem. This is where orthodoxy comes from. But the whole point of *any* model is to capture certain *essential* features of a system and to idealize away everything else. If the objective is computation, what could the essential features be, other than the computational primitives needed to perform efficiently? Efficient computations are more likely to be accurate than inefficient ones. Depending on what aspect one is interested in, one selects certain features to capture. A common misconception about models is that the models are supposed to *be* the system itself. But a model is an abstract way of reproducing what a system *does*. It's the same with this string transformation learning technique – the system represents abstractly *what happens* when molecular species encounter one another. What type of model is best will depend on what aspect of molecular behavior one is interested in. We do not have a single model to account for everything, just a single protocol for arriving at the models.

16

History and Epistemology

"The fundamental laws necessary for the mathematical treatment of a large part of physics and the whole of chemistry are thus completely known, and the difficulty lies only in the fact that application of these laws leads to equations that are too complex to be solved." – Paul Dirac, 1929.

This well-meaning, but unfortunate, statement of Dirac's clearly expresses the belief held by most scientists to this day that computational primitives equate with the fundamental parts of a system. If this statement could be erased from history and rewritten, I would write it so:

"The fundamental laws necessary for the mathematical *understanding* of a large part of physics and the whole of chemistry are thus completely known, and the difficulty lies only in the fact that *direct* application of these laws leads to equations that are too complex to be solved."

There is an old adage that claims that a whole is *greater* than the sum of its parts. What that actually means is that a whole is *different* than a straightforward sum of its parts. But what one finds in reality, computationally speaking, is that wholes are often *less* than the sum of their parts. For example, the Earth travels about the Sun in a fairly simple trajectory determined by a simple equation. We do not have to solve for the quantum mechanical orbits of each subatomic particle comprising the system.

I ask the question, is it actually necessary to use very detailed quantum mechanical calculations to predict the outcome of a chemical reaction? In light of what we were taught, the

way we were trained to think, the question almost seems ridiculous to pose, but is it? Considering the cost to human well-being of the *limitations* of current theoretical chemistry, it seems worth pondering. Just as an astrophysicist would not calculate the orbit of the Moon around the Earth by taking into account the quantum mechanical orbits of each subatomic particle composing it, to what degree can chemists do the same? Seems we should not be able to get far – "the molecules are more proximal to the quantum mechanical scale and thus ought to be more tightly ruled by it", one might say. However that is false reasoning because the Moon *is* just as tightly ruled by quantum mechanics as are electrons, so says the correspondence principle. Quantum mechanics is correct at any length scale we have been able to probe, classical physics being more an approximate statement of the properties of quantum mechanics in the limit of large numbers. Classical physics is an *epiphenomenon* – a secondary phenomenon that occurs alongside or in parallel to a primary phenomenon – quantum physics, in this case. Thus, we cannot conclude that simplifying epiphenomena (side effects) do not exist for molecular objects as well – epiphenomena unrelated to scale, but more related to the attainment of stable intermolecular states. Further, when I was an organic chemistry student many years ago, I was at that point in time not knowledgable in the field of quantum mechanics, yet I was able to predict the outcomes of thousands of organic reactions. How was I able to learn to do that? Thus, the epiphenomena, the rules of chemistry, must exist, it's just that we humans are having difficulty articulating them in an explicit way. String transformation learning solves that problem.

I want to state a principle that I am certain is universally true – The Epiphenomenological Principle: All physical laws are secondary. Physical laws always change as we drill down into the infinitesimal without end, and out into infinity without end. Physical laws, in reality, are no more "real" than the laws describing the behavior of an ant colony or school of fish. Mathematical laws, on the other hand, are not epiphenomenological. The moral I am trying to relate here is that it is not at all inappropriate to look for ways to do chemistry outside of strict quantum mechanical approaches, which have serious limitations that adversely affect human welfare. I am *not* saying quantum mechanics isn't the best way to approach *some* problems. I don't want to be black-and-white about things here. But let's not exclude other facets of nature which could be very rewarding.

A theory that is a bit further away from reality than a more accurate theory, however is superior in its imagery, symbolism and its ability to rationalize scientific fact may retain its value in the community. For example, consider the famous rivalry between valence bond (VB) theory and molecular orbital (MO) theory. The two theories were developed at about the same time, but quickly diverged into rival schools that have competed, sometimes fervently, on charting the mental map and epistemology (knowledge) of chemistry. It has been argued over the years that VB theory is flawed, and that MO theory is superior for explaining things like the structure of benzene. However, VB theory is central to the chemist's basic concept of what a chemical bond is. The once held belief that theories should automatically be thrown away in situations such as this is incognizant of the fact that even MO theory is a bit far from reality itself, and could be replaced with a quantum electrodynamic theory of

the molecule, and so on. Each theory must be valued for the unique merits it has, assuming it has substantial unique merits, as VB theory does. There exist acceptable levels of mythology in science, and I would even go so far as to say that often, a sufficient degree of mythology is preferable. Consider the case of classical mechanics – a false, but delightful approximation. So I must ask, is any science permanent? In light of this, perhaps. But never permanently considered "ultimate truth".

Each scientific field has an admissible problem space (understandably), and an admissible solution space (unfortunately). Of the physical sciences, physics seems to have the most complete solution space, borrowing from mathematics to the point of defining physics as the mathematical science. The connection of chemistry and biology to mathematics is once and twice removed, resp. Biology borrows heavily from chemistry, which in turn borrows from physics. There is very little direct interaction between mathematics and chemistry or biology. I find this situation unacceptable. There should be a "chemical mathematics" as well as a "biological mathematics". These fields need a solution space of their own, defined in terms of mathematics in a direct fashion. So far, the greatest achievements in the sciences have been attained when one field borrows and applies advancements made in another field. In the molecular sciences, this has meant the application of physics. A direct route must be laid between mathematics and chemistry in particular.

New theories do not emerge from old theories (and the belief that we must use complex quantum mechanical codes to calculate chemical reaction outcomes is a theory). New theories require the reevaluation of old theories; it is the old theories which must be readjusted to the new theories.

SMILES Strings

Bonnar's algorithm, a machine learning algorithm, requires that we form an abstract model of chemical reactivity data and we then use that model to make predictions about the reactivity of novel compounds (which were not in the training set). The most important issue in any application of machine learning is the representation of information that is fed into the system and/or obtained from it. The representation chosen has to be adapted to the problem and solution space. We need a way to represent molecular structures that captures all of the structural facets of their connectivity, yet consists of linear strings of characters such that `Mathematica`'s machine learning function **`Predict`** can work with the representation. The requirements of our representation are satisfied by the SMILES system of specifying molecular structure. As we will see in the next section, the SMILES representation is also ideal for automatically generating novel molecules.

The *simplified molecular-input line-entry system* (SMILES) is specification in the form of a line notation for describing the structure of chemical species using ASCII strings. SMILES strings can be imported by most molecule editors for conversion back into two-dimensional drawings or three-dimensional models of the molecules.

The original SMILES specification was initiated by David Weininger at the USEPA Mid-Continent Ecology Division Laboratory in Duluth in the 1980s. Acknowledged for their parts in the early development were Gilman Veith and Rose Russo (USEPA) and Albert Leo and Corwin Hansh (Pomona College) for supporting the work, and Arthur Weininger (Pomona, Daylight CIS) and Jeremy Scofield (Cedar River

Software, Renton, WA) for assistance in programming the system. The EPA funded the initial project to develop SMILES.

SMILES contains the same information as might be found in an extended connection table. The primary reason SMILES is more useful than a connection table is that it is a linguistic construct, rather than a computer data structure. SMILES is a true language, albeit with a simple vocabulary and only a few grammar rules. SMILES representations of structure can in turn be used as "words" in the vocabulary of other languages designed for storage of chemical information and chemical intelligence.

Part of the power of SMILES is that unique SMILES exist. With standard SMILES, the name of a molecule is synonymous with its structure; with unique SMILES, the name is universal. Anyone in the world who uses unique SMILES to name a molecule will chose the exact same name.

Canonicalization

SMILES denotes a molecular structure as a graph with optional chiral indications. This is essentially the two-dimensional picture chemists draw to describe a molecule. SMILES describing only the labeled molecular graph (i.e., atoms and bonds, but no chiral or isotopic information) are known as generic SMILES. There are usually a large number of valid generic SMILES which represent a given structure. A canonicalization algorithm exists to generate one special generic SMILES among all valid possibilities; this special one is known as the "unique SMILES". SMILES written with isotopic and chiral specifications are collectively known

as "isomeric SMILES". A unique isomeric SMILES is known as an "absolute SMILES". The following table gives some examples:

Input SMILES	Unique SMILES
OCC	CCO
[CH3][CH2][OH]	CCO
C-C-O	CCO
C(O)C	CCO
OC(=O)C(Br)(Cl)N	NC(Cl)(Br)C(=O)O
ClC(Br)(N)C(=O)O	NC(Cl)(Br)C(=O)O

SMILES Specification Rules

SMILES notation consists of a series of characters containing no spaces. Hydrogen atoms may be omitted (hydrogen-suppressed graphs) or included (hydrogen-complete graphs). Aromatic structures may be specified directly or in Kekulé form.

There are five generic SMILES encoding rules, corresponding to specification of atoms, bonds, branches, ring closures, and disconnections. Rules for specifying various kinds of isomerism also exist.

Atoms

Atoms are represented by their atomic symbols: this is the only required use of letters in SMILES. Each non-hydrogen atom is specified independently by its atomic symbol enclosed in square brackets, `[ ]`. The second letter of two-character symbols must be entered in lower case. Elements in the "organic subset" `B`, `C`, `N`, `O`, `P`, `S`, `F`, `Cl`, `Br`, and `I` may be written without brackets if the number of attached

hydrogens conforms to the lowest normal valence consistent with explicit bonds. "Lowest normal valences" are `B`(3), `C`(4), `N`(3,5), `O`(2), `P`(3,5), `S`(2,4,6), and 1 for the halogens. Atoms in aromatic rings are specified by lower case letters, e.g., aliphatic carbon is represented by the capital letter `C`, aromatic carbon by lower case `c`. Since attached hydrogens are implied in the absence of brackets, the following atomic symbols are valid SMILES notations:

`C`	methane	CH4
`P`	phosphine	PH3
`N`	ammonia	NH3
`S`	hydrogen sulfide	H2S
`O`	water	H2O
`Cl`	hydrochloric acid	HCl

Atoms with valences other than "normal" and elements not in the organic subset must be described in brackets.

`[S]`	elemental sulfur
`[Au]`	elemental gold

Within brackets, any attached hydrogens and formal charges must always be specified. The number of attached hydrogens is shown by the symbol `H` followed by an optional digit. Similarly, a formal charge is shown by one of the symbols `+` or `-`, followed by an optional digit. If unspecified, the number of attached hydrogens and charge are assumed to be zero for an atom inside brackets. Constructions of the form `[Fe+++]` are synonymous with the form `[Fe+3]`.

Examples are:

[H+]	proton
[Fe+2]	iron (II) cation
[Fe++]	iron (II) cation
[OH-]	hydroxyl anion
[OH3+]	hydronium cation
[NH4+]	ammonium cation

Bonds

Single, double, triple and aromatic bonds are represented by the symbols **-**, **=**, **#** and **:**, resp. Adjacent atoms are assumed to be connected to each other by a single or aromatic bond (atoms involved in aromatic bonds being distinguished by the fact their designated by lower case letters). Single and aromatic bonds may always be omitted. Examples are:

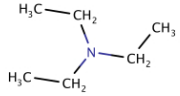
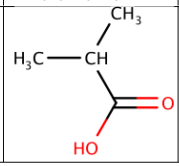
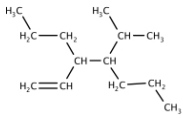
CC	ethane	CH3CH3
C=O	formaldehyde	CH2O
C=C	ethene	CH2=CH2
O=C=O	carbon dioxide	CO2
COC	dimethyl ether	CH3OCH3
C#N	hydrogen cyanide	HCN
CCO	ethanol	CH3CH2OH
[H][H]	molecular hydrogen	H2

For linear structure, SMILES notation corresponds to conventional diagrammatic notation except that hydrogens and single bonds are generally omitted. For example, 6-hydroxy-1,4-hexadiene can be represented by many equally valid SMILES, including the following three:

Structure	Valid SMILES
	C=CCC=CCO
CH2=CH-CH2-CH=CH=CH-CH2-OH	C=C-C-C=C-C-O
	OCC=CCC=C

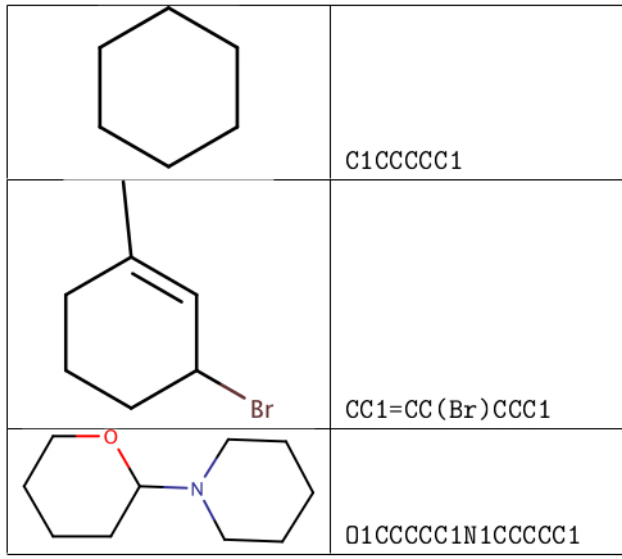
Branches

Branches are specified by enclosing them in parentheses, and can be nested or stacked. In all cases, the implicit connection to a parenthesized expression is to the left. Examples are:

Triethylamine	Isobutyric acid	3-propyl-4-isopropyl-1-heptene
CCN(CC)CC	CC(C)C(=O)O	C=CC(CCC)C(C(C)C)CCC
		

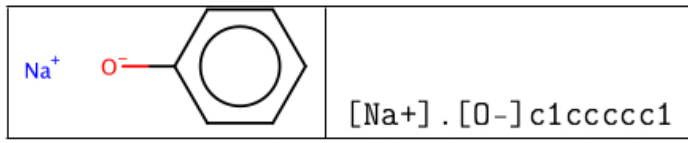
Cyclic Structures

Cyclic structures are represented by breaking one bond in each ring. The bonds are numbered in any order, designating ring opening (or ring closure) bonds by a digit immediately following the atomic symbol at each ring closure. Cyclohexane is an easy example.



Disconnected Structures

Disconnected compounds are written as individual structures separated by a "." (period). The order in which ions or ligands are listed is arbitrary. There is no implied pairing of one charge with another, nor is it necessary to have a zero net charge. If desired, the SMILES of one ion may be imbedded within another as shown in the example of sodium phenoxide:



Isomeric SMILES

This subsection describes the SMILES rules used to specify isotopism, configuration about double bonds, and chirality. The term *isomeric SMILES* collectively refers to SMILES written using these rules.

The SMILES isomer specification rules allow chirality to be completely specified for any structure, if it is known. Unlike most existing chemical nomenclatures such as CIP and IUPAC, these rules are also designed to allow rigorous partial specification of chirality. Aside from use in macros, substructure searching, and other pattern matching operations, this is important because much of the world's available chemical information is known for structures with incompletely resolved chiralities (not all possible chiral centers are separated, known, or reported).

All isomer specification rules in SMILES are therefore optional. The absence of a specification for any attribute implies that the value of that attribute is unspecified.

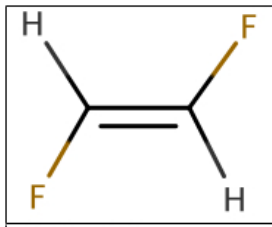
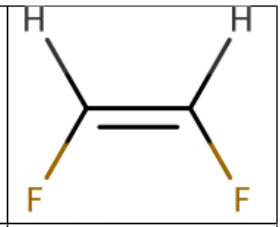
Isotopic Specification

Isotopic specifications are indicated by preceding the atomic symbol with a number equal to the desired integral atomic mass. An atomic mass can only be specified inside brackets. For instance:

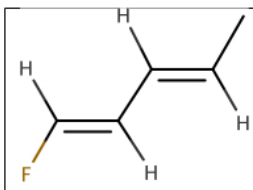
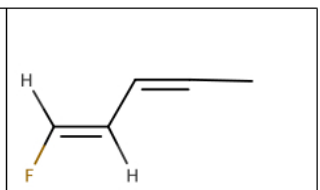
SMILES	Name
`[12C]`	carbon-12
`[13C]`	carbon-13
`[C]`	carbon (unspecified mass)
`[13CH4]`	C-13 methane

Configuration Around Double Bonds

Configuration around double bonds is specified by the characters `/` and `\` which are "directional bonds" and can be thought of as kinds of single or aromatic bonds. These symbols indicate relative directionality between the connected atoms, and have meaning only when they occur on both atoms which are double bonded. For instance, the following SMILES are valid for E- and Z-1,2-difluoroethene:

	
`F/C=C/F`	`F/C=C\F`
`F\C=C\F`	`F\C=C/F`

An important difference between SMILES chirality conventions and others such as CIP is that SMILES uses a local chirality representation (as opposed to absolute chirality), which allows partial specifications. An example of this is illustrated here:

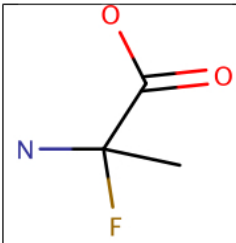
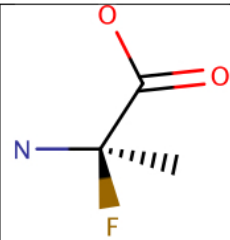
	
`F/C=C/C=C/C`	`F/C=C/C=CC`
completely specified	partially specified

Configuration Around Tetrahedral Centers

SMILES uses a very general type of chirality specification based on local chirality. Instead of using a rule-based numbering scheme to order neighbor atoms of a chiral center, orientations are based on the order in which neighbors occur in the SMILES string. As with all other aspects of SMILES, any valid order is acceptable.

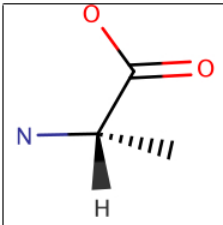
The simplest and most common kind of chirality is tetrahedral; four neighbor atoms are evenly arranged about a central atom, known as the "chiral center". If all four neighbors are different from each other in any way, mirror images of the structure will not be identical. The two mirror images are known as "enantiomers" and are the only two forms that a tetrahedral center can have. If two (or more) of the four neighbors are identical to each other, the central atom will not be chiral (its mirror images can be superimposed in space).

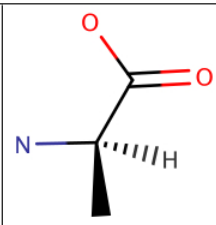
In SMILES, tetrahedral centers may be indicated by a simplified chiral specification (`@` or `@@`) written as an atomic property following the atomic symbol of the chiral atom. If a chiral specification is not present for a chiral atom, its chirality is implicitly not specified. For instance:

	
`NC(C)(F)C(=O)O`	`N[C@](C)(F)C(=O)O`
`NC(F)(C)C(=O)O`	`N[C@@](F)(C)C(=O)O`
unspecified chirality	specified chirality

Looking from the amino N to the chiral C (as the SMILES is written), the three other neighbors appear anticlockwise in the order that they are written in the top SMILES, and clockwise in the bottom one. The symbol `@` indicates that the following neighbors are listed anticlockwise, whereas `@@` indicates that the neighbors are listed clockwise.

If the central carbon is not the very first atom in the SMILES and has an implicit hydrogen attached (it can have at most one and still be chiral), the implicit hydrogen is taken to be the first neighbor atom of the three neighbors that follow a tetrahedral specification. If the central carbon is first in the SMILES, the implicit hydrogen is taken to be the "from" atom. Hydrogens may always be written explicitly (as `[H]`) in which case they are treated like any other atom. In each case, the implied order is exactly as written in SMILES. For example, some of the valid SMILES for alanine are:





`N[C@@]([H])(C)C(=O)O`	`N[C@]([H])(C)C(=O)O`
`N[C@@H](C)C(=O)O`	`N[C@H](C)C(=O)O`
`N[C@H](C(=O)O)C`	`N[C@@H](C(=O)O)C`
`[H][C@](N)(C)C(=O)O`	`[H][C@@](N)(C)C(=O)O`
`[C@H](N)(C)C(=O)O`	`[C@@H](N)(C)C(=O)O`

There are many kinds of chirality other than tetrahedral. More advanced chiral specifications are possible with SMILES but will not be covered here.

SMILES Conventions

Aside from the above rules, a small number of conventions are universally used in SMILES. These are briefly discussed below.

Hydrogens

Hydrogen atoms do not normally need to be specified when writing SMILES for most organic structures. The presence of hydrogens may be specified in three ways:

- Implicitly . . . for atoms specified without brackets, from normal valence assumptions.
- Explicitly by count . . . inside brackets, by hydrogen count supplied; zero if unspecified.
- As explicit atoms . . . as `[H]` atoms.

There are four situations where specification of explicit hydrogens are required:

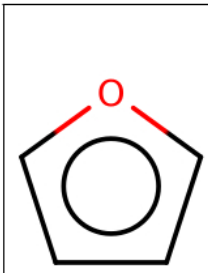
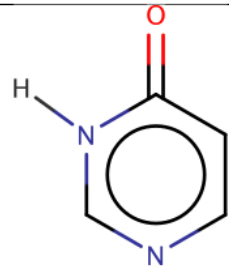
- charged hydrogen, i.e., a proton, `[H+]`
- hydrogens connected to other hydrogens, e.g., molecular hydrogen, `[H][H]`
- hydrogens connected to other than one other atom, e.g., bridging hydrogens
- isotopic hydrogen specifications,e.g., heavy water, `[2H]O[2H]`

Aromaticity

Aromaticity must be deduced in a system such as SMILES which generates an unambiguous chemical nomenclature because of the fundamental requirement to charaterize the symmetry of a molecule. Given effective aromaticity-detection algorithms, it is not necessary to enter any structure as aromatic if the user prefers to enter an aliphatic (Kekulé-like) structure. Entering structures as aromatic directly (i.e., by using lower case atomic symbols) provides a shortcut to accurate chemical specification and is closer to the mental molecular model used by most chemists.

The SMILES algorithm uses an extended version of Hueckel's rule to identify aromatic molecules and ions. To qualify as aromatic, all atoms in the ring must be sp^2-hybridized and the number of available "excess" p-electrons must satisfy Hueckel's 4N+2 criterion. As an example, benzene is written `c1ccccc1`, but an entry of `C1=CC=CC=C1` – cyclohexatriene, the Kekulé form – leads to detection of aromaticity and results in an internal structural conversion to aromatic representation. Conversely, entries of `c1ccc1` and `c1cccccc1` will produce the correct anti-aromatic structures for cyclobutadiene and cycleoctatetraene, `C1=CC=C1` and `C1=CC=CC=CC=C1`. In such cases the SMILES system looks for a structure that preserves the implied sp^2 hybridization, the implied hydrogen count, and the specified formal charge, if any. Some inputs, however, may not only be incorrect but also impossible, such as `c1cccc1`. Here `c1cccc1` cannot be converted to `C1=CCC=C1` since one of the carbon atoms would be sp^3 with two attached hydrogens. In such a structure alternating single and double bond assignments cannot be made. The SMILES system will flag this as an

"impossible" input. Note that only C, N, O, P, S, As, Se, and * (wildcard) atoms can be considered aromatic.

	
C1=COC=C1	C1=CN=C[NH]C(=O)1
c1cocc1	c1cnc[nH]c(=O)1

Extensions for Reactions

The SMILES language is extended to handle reactions. There are two areas where SMILES is extended: distinguishing component parts of reactions and atom maps. We will not discuss atom maps here.

Component parts of a reaction are handled by introducing the **>** character as a new separator. Any reaction must have exactly two **>** characters in it. Each of the **>**-separated components of a reaction must be a valid SMILES.

As an aside, molecule SMILES never have a **>** character. In a program, one can quickly determine if a SMILES refers to a reaction or molecule by searching for a **>** character in the string.

Examples of reaction SMILES are:

```
C=CCBr>>C=CCl
```

This is a valid SMILES reaction. Note that there are no agent molecules, which would appear between the two `>` symbols. Also note that several atoms are missing from the reaction.

```
[I-].[Na+].C=CCBr»[Na+].[Br-].C=CCl
```

This is a more complete version of the same reaction. It has been canonicalized.

```
C=CCBr.[Na+].[I-]>CC(=O)C>C=CCl.[Na+].[Br-]
```

This version of the reaction includes an agent. Note that the SMILES does not indicate how the agent participates. Whether the agent is a solvent, catalyst, or performs another function within the reaction must be stored separately as data.

SMILES Chemical Reaction Database

Bonnar's algorithm requires that we form an abstract model of chemical reactivity. We will compute a map from reactants to products. A chemical reaction database, the SMILES chemical reaction database, will be used for this purpose and the reader may download it from the webpage. The SMILES Chemical Reaction Database has two million entries and is contained in six files, **`rxnlist01.smiles`** through **`rxnlist06.smiles`**. All of the SMILES reaction strings in the database follow the pattern **`reactants»products`**, have no information regarding agents, and are separated by newline characters.

We will only be able to work with a small subset of the entire database. We will extract reactions that are bimolecular

and involve large reactants, such that the machine learning system learns something about the dynamics involved in transforming large reactants, as in a reaction involving the specific transformation of DNA. We will also concentrate on reactions involving only the organic subset of elements.

Some of these things are accomplished in the first **Mathematica** notebook we discuss, **rxnsfilemaker.nb**, which is in the notebooks directory. We shall discuss that notebook here. The first step is to set your active directory to where you have the database files stored (change the directory name to your own directory name):

```
SetDirectory["/home/james/Desktop/project/notebooks"]
/home/james/Desktop/project/notebooks
```

You can check your current directory as follows:

```
Directory[]
/home/james/Desktop/project/notebooks
```

You can see the files in the current directory as follows:

```
FileNames[]
{generate_markov_text.py, rxnlist01.smiles,
 rxnlist02.smiles, rxnlist03.smiles, rxnlist04.smiles,
 rxnlist05.smiles, rxnlist06.smiles, rxnsfilemaker.nb}
```

Next we import each of the SMILES database files as a **List**:

```
list1 = Import["rxnlist01.smiles", "List"];
list2 = Import["rxnlist02.smiles", "List"];
list3 = Import["rxnlist03.smiles", "List"];
list4 = Import["rxnlist04.smiles", "List"];
list5 = Import["rxnlist05.smiles", "List"];
list6 = Import["rxnlist06.smiles", "List"];
```

We join each of the lists into one large list:

```
list = list1 ~ Join ~ list2 ~ Join ~ list3 ~ Join ~ list4 ~ Join ~ list5 ~ Join ~ list6;
```

We can check the length of the list as follows:

```
Length[list]
```

```
2 000 000
```

Next we want to narrow down our database to only bimolecular reactions, e.g., of the form **A.B»C** having two reactants and one product (note that balanced equations are unnecessary in our work). The following code selects only the strings matching that pattern:

```
bimolecularlist =
  Select[list,
   StringMatchQ[#, Except["."] .. ~~ "." ~~ Except["."] .. ~~ ">>" ~~
      Except["."] ..] &];
```

Now checking the length of that list:

```
Length[bimolecularlist]
```

```
753 113
```

We're only interested in reactions exclusively involving atoms in the organic subset. The following code selects only reaction strings free of atoms not in the organic subset:

```
organiclist = Select[bimolecularlist,
   StringFreeQ[#, {"Rb", "Cs", "Fr", "Sr", "Ba", "Ra", "Sc", "Ti",
      "V", "Cr", "Mn", "Fe", "Co", "Ni", "Cu", "Zn", "Y", "Zr", "Nb",
      "Mo", "Tc", "Ru", "Rh", "Pd", "Ag", "Cd", "Lu", "Hf", "Ta",
      "W", "Re", "Os", "Ir", "Pt", "Au", "Hg", "Al", "Ga", "In", "Tl",
      "Si", "Ge", "Sn", "Pb", "As", "Sb", "Bi", "Se", "Te", "Po",
      "At", "He", "Ne", "Ar", "Kr", "Xe", "Rn", "La", "Ce", "Pr",
      "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho", "Er", "Tm",
      "Yb", "Ac", "Th", "Pa", "U", "Np", "Pu", "Am", "Cm", "Bk", "Cf",
      "Es", "Fm", "Md", "No"}] &];
```

Again checking the length of the list:

```
Length[organiclist]
```

```
618097
```

We are interested in reactions involving complex molecular species, large molecules with complex interactions. So the following code is used to select SMILES reaction strings having a length greater than 400:

```
bigorganiclist = Select[organiclist, StringLength[#] > 400 &];
```

Once again checking the length of the list:

```
Length[bigorganiclist]
```

```
3674
```

We shall be taking a hundred reactions from this list in another notebook, so to avoid taking reactions that are very similar and may have been proximal in the database, we randomize their order:

```
randomizedlist = RandomSample[bigorganiclist, Length[bigorganiclist]];
```

We can see what symbols are used in our list (our "alphabet") in the following way:

```
(*Delete[#,0]& replaces the Head with Sequence*)

alphabet = Union[Delete[Characters[randomizedlist], 0]]

{@, ¤, %, (, ), -, +, =, [, ], \, >, ., /, 0, 1, 2, 3, 4, 5, 6,
 7, 8, 9, a, B, c, C, F, g, H, I, K, l, M, n, N, o, O, P, r, s, S}
```

You may now reset the current directory to where you would like to export your list if you want. Here, the file name is **bigrxns.smiles**, The entries in the file are separated by newline characters. The entire file can then be opened in **MarvinView** or individual entries can be copied and pasted into **Chemicalize**.

```
Export["bigrxns.smiles", randomizedlist, "List"];
```

Markovian Text Generator

In this section it is shown how to use a Markovian text generator to generate pseudo-random SMILES strings which represent an in silico library of candidate molecules for testing using Bonnar's algorithm. These pseudo-random molecules follow a theme yet vary somewhat in structure.

In the 1948 landmark paper *A Mathematical Theory of Communication*, Claude Shannon founded the field of information theory and revolutionized the telecommunications industry, laying the groundwork for the Information Age. In this paper, Shannon proposed using a Markov chain to create a statistical model of the sequences of letters in a piece of text. Markov chains are now widely used in machine recognition, information retrieval, data compression, and spam filtering. They also have many scientific computing applications including: the genemark algorithm for gene prediction, the Metropolis algorithm for measuring thermodynamic proper-

ties, and Google's pagerank algorithm for web search, and last but not least, generating pseudo-random text.

Shannon approximated the statistical structure of a piece of text using a simple mathematical model known as a *Markov model*. A Markov model of order 0 predicts that each letter in the alphabet occurs with a fixed probability. We can fit a Markov model of order 0 to a specific piece of text by counting the number of occurrences of each letter in that text, and using these counts as probabilities. For example, if the input text is "`agggcagcgggcg`", then the Markov model of order 0 predicts that each letter is `a` with probability 2/13, `c` with probability 3/13, and `g` with probability 8/13. The following sequence of letters is a typical example generated from this model:

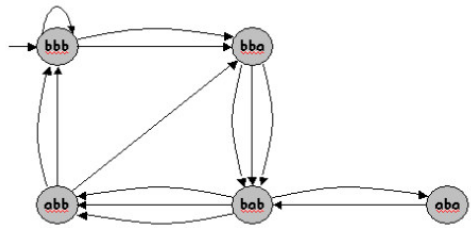
```
a g g c g a g g g a g c g g c a g g g g ...
```

An order 0 model assumes that each letter is chosen independently. This does coincide with the statistical properties of English text, for instance, since there is a high correlation among successive letters in an English word or sentence. For example, the letters `h` and `r` are much more likely to follow `t` than either `c` or `x`.

We obtain a more refined model by allowing the probability of choosing each successive letter to depend on the preceding letter or letters. A *Markov model of order n* predicts that each letter occurs with fixed probability, but that probability depends on the previous n consecutive letters (n-gram). For example, if the text has 100 occurrences of `th`, with 60 occurrences of `the`, 25 occurrences of `thi`, 10 occurrences of `tha` and 5 occurrences of `tho`, the Markov model of order

2 predicts that the next letter following the 2-gram **th** is **e** with probability 3/5, **i** with probability 1/4, **a** with probability 1/10 and **o** with probability 1/20.

An approach to generate text according to a Markov model is to create a "Markov chain" and simulate a trajectory through it. A Markov chain is comprised of a set of states, one distinguished state called the start state, and a set of transitions from one state to another. The figure below illustrates a Markov chain with 5 states and 14 transitions.



To simulate a trajectory through the Markov chain, begin at the start state. At each step select one of the leaving arcs uniformly at random, and move to the neighboring state. For example, if the Markov chain is in state **bab**, then it will transition to state **abb** with probability 3/4 and to state **aba** with probability 1/4. The following are the first seven steps of a possible trajectory beginning from **bbb**:

bbb→**bbb**→**bba**→**bab**→**abb**→**bbb**→**bba**

A pseudo-random text can be generated according to a Markov model of order n by building a Markov chain froma piece of text, and simulating a trajectory through the Markov chain. A state is included for each of the distinct n-grams that occur in the text. N transitions are included, one from

each n-gram in the text to the next overlapping n-gram. Text is treated as a cyclic string to handle boundary cases. For example, if $n = 3$ and the text consists of **bbbabbabbbbaba**, then the first two transitions to add are from **bbb** to **bba** and from **bba** to **bab**, and the last (cyclic) one is from **abb** to **bbb**. The full Markov chain for $n = 3$ is illustrated in the previous figure.

All of these ideas are implemented in the python script **generate_markov_text.py**, which can be downloaded from my webpage on internet archive. This script generates pseudo-random text using a Markov chain model. The Markov chain encodes the statistical properties of a source text with a user-specified n-gram length. Note that the script treats spaces and newline characters as characters.

The script can be used with two kinds of sources:

1. A text file from which the Markov chain is built.
2. A text file with a pre-generated Markov chain from a previous run.

The generated text can be printed on the console or written to a file, while the Markov chain can be exported to a file and used later (the source file is no longer needed).

Usage examples:

```
$python generate_markov_text -s input_text.txt -l
7 -n 1000 -o output_file.txt -w markov.txt
```

generates 1000 letters using a 7-gram, writes the generated text to **output_file.txt** and writes the Markov chain to **markov.txt**.

```
$python generate_markov_text -r markov.txt -n 1000
-c
```

reads a pre-generated Markov chain from a file and prints 1000 letters on the console.

SMILES String Generation

Generating pseudo-random SMILES strings that describe molecules (that follow a theme loosely) is a straignhtforward application of the Markovian text generator. The only complication is the generation of invalid SMILES strings, which are mainly due to mismatched brackets and parentheses. We simply filter out those occurrences. The Mathematica notebook **moleculesSourceAndGeneterator.nb** gives the basics of how to generate SMILES strings using the Markovian text generator.

First, set the current directory to where the source file, **source.smiles** and the program **generate_markov_text.py** are located.

```
SetDirectory["/home/james/Desktop/project/notebooks"]
```

```
/home/james/Desktop/project/notebooks
```

```
FileNames[]
```

```
{balancedparentheses.nb, bigrxns.smiles, DNAsmiles.nb,
 generate_markov_text.py, learnable.nb, moleculesSourceAndGenerator.nb,
 moleculesSource.nb, R175H.smiles, rxnlist01.smiles, rxnlist02.smiles,
 rxnlist03.smiles, rxnlist04.smiles, rxnlist05.smiles,
 rxnlist06.smiles, rxnsfilemaker.nb, source.smiles, tp53.smiles}
```

Next we import our reaction list:

```
rxnlist = Import["bigrxns.smiles", "List"];
```

```
Length[rxnlist]
```

```
3674
```

We split each expression at the » symbol. As far as this task is concerned, we are only interested in the reactants not the product.

```
rxnlist2 = StringSplit[#, ">>"] & /@ rxnlist;
```

```
reactionList3 = rxnlist2[[All, 1]]
```

```
{[H][C@@](C)(NC(=O)[C@](C)(CC(C)C)N(C)C(=O)[C@@](C)(NC(=O)[C@](C)(CC(C)C)N(
    C)C(=O)CN(C)C(=O)[C@](C)(CCC)NC(=O)[C@](C)(C...
  )C(=O)N[C@](C)(C)C(=O)N(C)[C@@](C)(CC(C)C)C(=O)N(C)[C@@](C)(CC(C)C)C(=O)
    N(C)[C@@](C)(C(C)C)C(=O)OC(C)(C)C.OC(=O)C(F)(F)F,
 ··· 3672 ···, ...}
```

large output | show less | show more | show all | set size limit...

These reactants will function as our source data for the Markovian text generator. The idea to to generate random molecules to react with R175H-mutant DNA in an effort to detect lead compounds for the correction of the gene.

```
moleculeslist = StringSplit[#, "."] & /@ reactionList3;
```

```
sourceList = moleculeslist // Flatten
```

```
{[H][C@@](C)(NC(=O)[C@](C)[H])(CC(C)C)N(C)C(=O)[C@@](C)[H])(NC(=O)[C@](C)[H])(CC(C)C)N(
    C)C(=O)CN(C)C(=O)[C@](C)[H])(CCC)NC(=O)[C@](C)[H] ...
  1ccccc1)C(C)C)C(=O)N[C@](C)[H])(C)C(=O)N(C)[C@@](C)[H])(CC(C)C)C(=O)N(C)[C@@](C)[H])
    (CC(C)C)C(=O)N(C)[C@@](C)[H])(C(C)C)C(=O)OC(C)(C)C, <<<7346>>>, ...}
```

large output | show less | show more | show all | set size limit...

```
sourceList[[5]]
```

```
[H][C@]1(C)O[C@@](C)[H])(OCC2\C=C(C)\C=C\C(=O)C(C)CC(CC=O)C(O[C@]3([H])O[C@](C)[H])(C)[C
  @@](C)[H])(OC(C)=O)[C@](C)[H])(N(C)C)[C@@]3([H])OC(C)=O)C(C)C(O)CC(=O)OC2CC)[C@](C)[H])(
  OC)[C@](C)[H])(OC)[C@@]1([H])O
```

```
sourceList = DeleteDuplicates[sourceList];
```

```
Length[sourceList]
```

```
3914
```

We can export and import the source.

```
Export["source.smiles", sourceList, "List"]
```

```
source.smiles
```

```
sourceList = Import["source.smiles", "List"]
```

```
{[H][C@@](C)(NC(=O)[C@](C)[H])(CC(C)C)N(C)C(=O)[C@@](C)[H])(NC(=O)[C@](C)[H])(CC(C)C)N(
    C)C(=O)CN(C)C(=O)[C@](C)[H])(CCC)NC(=O)[C@](C)[H] ...
  1ccccc1)C(C)C)C(=O)N[C@](C)[H])(C)C(=O)N(C)[C@@](C)[H])(CC(C)C)C(=O)N(C)[C@@](C)[H])
    (CC(C)C)C(=O)N(C)[C@@](C)[H])(C(C)C)C(=O)OC(C)(C)C, <<<3912>>>, ...}
```

large output | show less | show more | show all | set size limit...

We need to declare some functions for culling the generated SMILES strings which have mismatched brackets and parentheses.

```
balancedParentheses[string_String] := Module[{mystring, parenList, stackList},
  mystring = {string};
  parenList = Characters[StringDelete[mystring, Except[{"(", ")"}]]][[1]];
  stackList = {};
  For[i = 1, i ≤ Length[parenList],
   i++, If[parenList[[i]] === "(", AppendTo[stackList, 1],
    If[Length[stackList] > 0, stackList = Drop[stackList, -1]]]];
  If[Length[stackList] === 0 && StringCount[string, "("] ===
     StringCount[string, ")"], mystring[[1]], "MISMATCHES"]
 ]
```

```
mylist = {"a(b)()(())", "(c()", "(a)(d)",
   ")(", "()(b)", ")))(", "[[(a]b[c]])", "([a)[b[]", "]]()][";
```

```
balancedParentheses[#] & /@ mylist
```

```
{a(b)()(()), MISMATCHES, (a)(d), MISMATCHES,
 ()(b), MISMATCHES, [[(a]b[c]]), ([a)[b[], ]]()][}
```

```
mylist2 = {"))", ")()", "())", "))((", "))(()))", ")(", "))"};
```

```
balancedParentheses[#] & /@ mylist2
```

```
{MISMATCHES, MISMATCHES, MISMATCHES, MISMATCHES, MISMATCHES, MISMATCHES, MISMATCHES}
```

```
takeBalancedParentheses[initialList_List] := Module[{finalList, tempList},
  tempList = balancedParentheses[#] & /@ initialList;
  finalList = DeleteCases[tempList, "MISMATCHES"]
 ]
```

```
list1 = takeBalancedParentheses[mylist]
```

```
{a(b)()(()), (a)(d), ()(b), [[(a]b[c]]), ([a)[b[], ]]()][}
```

```
balancedBrackets[string_String] := Module[{mystring, bracketList, stackList},
  mystring = {string};
  bracketList = Characters[StringDelete[mystring, Except[{"[", "]"}]]][[1]];
  stackList = {};
  For[i = 1, i ≤ Length[bracketList],
   i++, If[bracketList[[i]] === "[", AppendTo[stackList, 1],
    If[Length[stackList] > 0, stackList = Drop[stackList, -1]]]];
  If[Length[stackList] === 0 && StringCount[string, "["] === StringCount[string, "]"],
   mystring[[1]], "MISMATCHES"]
 ]
```

```
takeBalancedBrackets[initialList_List] := Module[{finalList, tempList},
  tempList = balancedBrackets[#] & /@ initialList;
  finalList = DeleteCases[tempList, "MISMATCHES"]
 ]
```

```
list3 = takeBalancedBrackets[list1]
```

```
{a(b)()(()), (a)(d), ()(b), [[(a]b[c]])}
```

```
balancedList[list_List] := takeBalancedBrackets[takeBalancedParentheses[list]]
```

```
balancedList[mylist]
```

```
{a(b)()(()), (a)(d), ()(b), [[(a]b[c]])}
```

```
(*Run Markov generator*)

Run["python generate_markov_text.py -s source.smiles
    -l 20 -n 1000000 -o generated.smiles -w markov.txt"]

0

listA = Import["generated.smiles", "List"];

Length[listA]

5750

listA = DeleteDuplicates[listA];

Length[listA]

5257

listB = balancedList[listA];
```

```
Length[listB]
1835

Export["generated.smiles", listB, "List"];

Length[sourceList]
3914

testlist = sourceList~Join~listB;

Length[testlist]
5749

testlist = DeleteDuplicates[testlist];

Length[testlist]
5286
```

Towards a Cure for R175H-Positive Cancer

What exactly would it mean to *cure* cancer? To *cure* this disease, means to remedy its cause rather than its symptoms, further to transform the faulty gene responsible for the disease to the healthy variant – to correct the DNA sequence. How are we to accomplish that? An agent must be found that is small enough to enter the cell and nucleus, that is impervious to the action of nucleases, that does not illicit an immune response, that can selectively recognize the R175H DNA sequence, and changes the a-t base pair at precisely the right position to a g-c base pair in the correct orientation.

The DNA sequences we shall use in the application of Bonnar's algorithm are given in the notebook **DNAsmiles.nb** and the files giving their SMILES strings are **tp53.smiles** and **R175H.smiles**. We list the contents of the notebook here:

```
SetDirectory["/home/james/Desktop/project/notebooks"]
```
```
/home/james/Desktop/project/notebooks
```

```
dnaTP53subsequence =
  "cacatgacggaggttgtgaagcgctgcccccaccatgagcgctgcgcagcgctcatggtgggggcagcgcttcacaac\
     ctccgtcatgtg";
```

```
dnaR175Hsubsequence =
  "cacatgacggaggttgtgaagcactgcccccaccatgagcgctgcgcagcgctcatggtgggggcagtgcttcacaac\
     ctccgtcatgtg";
```

```
StringLength[dnaTP53subsequence]
```
```
90
```

```
dnaTP53List = Characters[dnaTP53subsequence]
```
```
{c, a, c, a, t, g, a, c, g, g, a, g, g, t, t, g, t, g, a, a, g, c, g, c, t, g, c, c, c, c, c, a,
 c, c, a, t, g, a, g, c, g, c, t, g, c, g, c, a, g, c, g, c, t, c, a, t, g, g, t, g, g,
 g, g, g, c, a, g, c, g, c, t, t, c, a, c, a, a, c, c, t, c, c, g, t, c, a, t, g, t, g}
```

```
TP53smiles =
  dnaTP53List /. {"a" → "OP([O-])(=O)OC[C@H]1O[C@@H](n2cnc3c2ncnc3(N))C[C@@H]1",
                  "c" → "OP([O-])(=O)OC[C@H]1O[C@@H](n2ccc(N)nc2(=O))C[C@@H]1",

    "g" → "OP([O-])(=O)OC[C@H]1O[C@@H](n2cnc3c2nc(N)[nH]c3(=O))C[C@@H]1",
                "t" →
     "OP([O-])(=O)OC[C@H]1O[C@@H](n2cc(C)c(=O)[nH]c2(=O))C[C@@H]1"};
```

```
TP53smiles[[1]]
```
```
OP([O-])(=O)OC[C@H]1O[C@@H](n2ccc(N)nc2(=O))C[C@@H]1
```

```
TP53smiles[[1]] = "[-O]P([O-])(=O)OC[C@H]1O[C@@H](n2ccc(N)nc2(=O))C[C@@H]1"
```
```
[-O]P([O-])(=O)OC[C@H]1O[C@@H](n2ccc(N)nc2(=O))C[C@@H]1
```

```
Length[TP53smiles]
```
```
90
```

```
TP53smiles[[90]]
```
```
OP([O-])(=O)OC[C@H]1O[C@@H](n2cnc3c2nc(N)[nH]c3(=O))C[C@@H]1
```

```
TP53smiles[[90]] =
 "OP([O-])(=O)OC[C@H]1O[C@@H](n2cnc3c2nc(N)[nH]c3(=O))C[C@@H]1OP([O-])([O-])=O"
```
```
OP([O-])(=O)OC[C@H]1O[C@@H](n2cnc3c2nc(N)[nH]c3(=O))C[C@@H]1OP([O-])([O-])=O
```

```
TP53string = StringJoin[TP53smiles];

Export["tp53.smiles", TP53string, "List"]
```

tp53.smiles

```
dnaR175HList = Characters[dnaR175Hsubsequence]
```

```
{c, a, c, a, t, g, a, c, g, g, a, g, g, t, t, g, t, g, a, a, g, c, a, c, t, g, c, c, c, c, c, a,
 c, c, a, t, g, a, g, c, g, c, t, g, c, g, c, a, g, c, g, c, t, c, a, t, g, g, t, g, g,
 g, g, g, c, a, g, t, g, c, t, t, c, a, c, a, a, c, c, t, c, c, g, t, c, a, t, g, t, g}
```

```
R175Hsmiles =
  dnaR175HList /. {"a" → "OP([O-])(=O)OC[C@H]1O[C@@H](n2cnc3c2ncnc3(N))C[C@@H]1",
                   "c" → "OP([O-])(=O)OC[C@H]1O[C@@H](n2ccc(N)nc2(=O))C[C@@H]1",

    "g" → "OP([O-])(=O)OC[C@H]1O[C@@H](n2cnc3c2nc(N)[nH]c3(=O))C[C@@H]1",
                  "t" →
     "OP([O-])(=O)OC[C@H]1O[C@@H](n2cc(C)c(=O)[nH]c2(=O))C[C@@H]1"};
```

```
R175Hsmiles[[1]] = "[-O]P([O-])(=O)OC[C@H]1O[C@@H](n2ccc(N)nc2(=O))C[C@@H]1"
```

[-O]P([O-])(=O)OC[C@H]1O[C@@H](n2ccc(N)nc2(=O))C[C@@H]1

```
R175Hsmiles[[90]] =
 "OP([O-])(=O)OC[C@H]1O[C@@H](n2cnc3c2nc(N)[nH]c3(=O))C[C@@H]1OP([O-])([O-])=O"
```

OP([O-])(=O)OC[C@H]1O[C@@H](n2cnc3c2nc(N)[nH]c3(=O))C[C@@H]1OP([O-])([O-])=O

```
R175Hstring = StringJoin[R175Hsmiles];

Export["R175H.smiles", R175Hstring, "List"]
```

R175H.smiles

It should be noted that we're using a 90-base-long subsequence that is complimentary to itself, i.e., hairpin DNA which forms a double helix even though it is single-stranded (such that we can describe the molecule with a single SMILES string). Three-dimensional structure is only treated in an implicit way – its effect on reactivity is learned.

Bonnar's Algorithm